

Program 3 solution outline

see companion files in bgunix
cs3350 lib: p3soln

suggested approach

1. get the file I/O working.
 - read and echo lines from the file
2. Convert to upper case
3. parse each line to extract 'valid' words
4. Create `IndexItem` class to store:
 - word
 - its list of line numbers
5. Use `list<indexItem>` in main to handle list

Step 1. file I/O

choice: read the file

char by char

OR

whole line at a time using `getline`

either approach can work

read/echo the file

```
string aLine;
ifstream inf("p3.txt");
int lineCount = 1;
while ( getline(inf, aLine) ) {
    cout << setw(4) << lineCount
         << " " << fline << endl;
    lineCount++;
}
```

See file: v1.cpp in p3soln folder

Step 2. Convert to LowerCase

```
void convertToLower( string & aLine) {  
    for ( int i=0; i < aLine.size(); i++) {  
        if ( aLine[i] >= 'A' && aLine[i] <= 'Z' )  
            aLine[i] += 'a' - 'A';  
    }  
}  
  
See v2.cpp
```

Step 3. parse a line of text

Choices:

- String find variations:
 - find, findfirstof, findfirstnotof
- State machine
- JS approach
- See v3a.cpp , v3b.cpp

parsing a line - caution

```
i = 0;  
while ( fline[i] != NULL ) {  
    if ( fline[i] isValid ) {  
        j = 0;  
        while ( fline[j] isValid ) j++;  
        newword = fline(i..j-1);  
        i = j (j+1?);  
    }  
    else i++;  
}  
  
ISSUE: nesting the j while loop can cause logic  
headaches
```

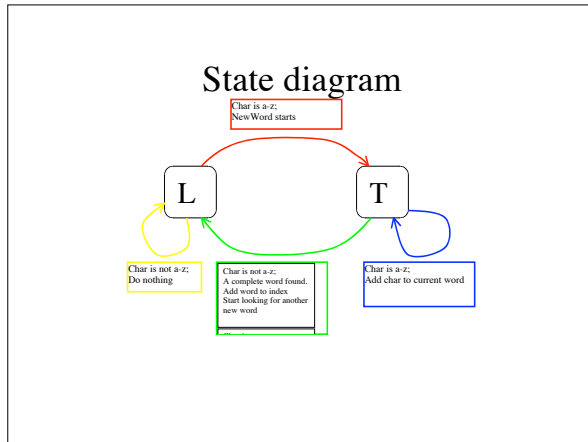
parsing the line:

state machine approach

Two states:

- 1) looking for the beginning of a word (L)
- 2) collecting word characters (T)

that is: we look at each char from the line in turn and process it according to what state we are in.



translation of state diagram into C++

```

state = 'L'; // start with looking state.
for (i=0; i < lineLength; i++) {
  ch = line[i]; // ith character
  switch (state) {
  case 'L':
    if ch is a-z then // beginning of a word
      newword = ch
      state = 'T'
    else; // nothing
  case 'T':
    if ch is a-z then
      newword += ch ; // add char to word
    else // terminating char, newword complete
      add word to index
      state = 'L'; // go back to looking
  } // end switch
} // end for loop
  
```

See: v3b.cpp in p3soln

Step 4. IndexItem class

```

class IndexItem {
private:
  string iword;
  list<int> linenumbers;
public:
  //constructor
  // accessor functions

  Since this is an 'embedded' class, making everything
  public would be 'easier'
  See indexitem.h, indexitem.cpp in p3soln
  
```

Step 5. employ IndexItem with STL list container

```

list<IndexItem> index;
// this is the index of words

for each new word found, we check to see if
the word is already in the index.
if so, update the linenumbers list for that
word
if not, add the word to the index
  
```

Utilities 1:

```
// is target word in the list?
// return an iterator that points at:
// the word : if found
// the end of list: if not found

list<indexItem>::iterator
searchList( list<indexItem>::iterator first,
            list<indexItem>::iterator last,
            const string & target) {
list<indexItem>::iterator iter = first;
while ( iter != last &&
        ( (*iter).getWord() != target)) iter++;

return iter;
}
```

Utilities 1a: break

```
// is target word in the list?
// return an iterator that points at:
// the word : if found
// the end of list: if not found

list<indexItem>::iterator
searchList( list<indexItem>::iterator first,
            list<indexItem>::iterator last,
            const string & target) {
list<indexItem>::iterator iter = first;
for ( , iter != last, iter++)
    if ( (*iter).getWord() == target))
        break; // out of loop

return iter;
}
```

Utilities 2:

```
void addWordToIndex( list<indexItem> & index,
                    string & word, int lineNum) {
list<indexItem>::iterator res;

res = searchList(index.begin(),index.end(), word);

if ( res == index.end() ) {not in list. add new
indexItem nii(word,lineNum);
index.push_back(nii);
index.sort();
}
else { // already in list. add line number
(*res).addLineNumber(lineNum);
}
}
```

v4.cpp

- this file combines the results of the 4 steps into a solution for the problem.
- to compile: g++ v4.cpp indexitem.cpp